Applying a Multi-Paradigm Approach to Implementing Wireless Sensor Network Based River Monitoring

Jo Ueyama
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo (USP)
13566-585 São Carlos, SP, Brazil
joueyama@icmc.usp.br

Danny Hughes, Ka Lok Man, Steven Guan
Computer Science and Software Engineering
Xi'an Jiaotong-Liverpool University
Suzhou, Jiangsu, 215123, China
{daniel.hughes, ka.man, steven.guan}@xjtlu.edu.cn

Nelson Matthys, Wouter Horré, Sam Michiels, Christophe Huygens, Wouter Joosen
IBBT-DistriNet, Katholieke Universiteit Leuven, Leuven, 3000, Belgium
{nelson.matthys, wouter.horre, sam.michiels,, christophe.huygens, wouter.joosen}@cs.kuleuven.be

***Abstract*—This paper describes the application of the DisSeNT middleware to implement Wireless Sensor Network based river monitoring. DisSeNT provides LooCI, an efficient run-time reconfigurable component model, PMA, a lightweight policy-based management framework and QARI, a declarative quality-aware deployment framework. Using a river monitoring case-study, this paper analyses how these distinct software development paradigms can be used in a complimentary fashion to develop efficient wireless sensor network applications. The resulting system has been deployed and evaluated in a real-world river monitoring scenario in the city of São Carlos, Brazil.**

Keywords-component; Wireless Sensor Networks, Multi-Paradigm Programming, Middlewar.

I. Introduction

Wireless Sensor Networks (WSN) are a promising platform for supporting environmental monitoring. WSN are composed of tiny embedded computers known as 'motes' that are equipped with low-power wireless networking technologies and simple sensors. These motes form ad-hoc, self-organizing networks that are capable of sensing and reacting to the physical environment. WSN promise a number of advantages for environmental monitoring compared to traditional telemetry systems. These include: (i.) reduced cost, (ii.) simplified deployment, (iii.) high resolution sensing and (iv.) the ability to adapt to changing environmental conditions.

Software development for WSN is highly complex due to the inherent unreliability of low-power networking technologies, the tight resource constraints of embedded platforms and a tight-coupling with the environment that leads to a high degree of dynamism. To address these problems, the DistriNet Sensor Network Toolkit (DisSeNT) provides a suite of multiparadigm software engineering tools [1], [2], [3] that are capable of managing the complexities of WSN software development. This multi-paradigm approach is described in detail in [15]. This paper extends our previous work by applying this approach to develop an efficient, manageable and adaptable WSN-based river monitoring application.

The remainder of this document is structured as follows: Section 2 discusses related work. Section 3 provides a brief overview of the DisSeNT middleware toolkit. Section 4 provides a detailed description of the river monitoring application composition. Section 5 evaluates the DisSeNT approach. Finally, Section 6 discusses directions for future work and concludes.

II. Related Work

DisSeNT is unique in that it combines a number of programming paradigms to provide rich support for WSN application development. Existing work in the field of each supporting technology: component based software engineering, policy based management and quality aware software deployment is discussed in section A to C below.

A. *Component Models for Wireless Sensor Networks*

NesC [5] is perhaps the best known and most widely deployed component model for WSN. NesC provides an event-driven programming approach together with a static component model. NesC components cannot be dynamically rewired to support reconfiguration and adaptation. However, this static approach allows for whole program analysis and optimization. TinyOS provides no support for remote bindings, limiting the support provided for constructing complex distributed applications.

OpenCOM [6] and its embedded instantiation RUNES [7] are language undependent and feature a compact run-time kernel that supports dynamic reconfiguration.

OpenCOM also offers a higher level of abstraction, known as Component Frameworks (CFs), which are used to model interactions between cooperating components. In the case of distributed component frameworks, a Meta Object Protocol (MOP) allows reconfiguration actions to be applied to groups of components. While OpenCOM notionally supports diverse binding types, all current instantiations use RPC bindings which, as argued in our previous work are too heavyweight and tightly-coupled for WSN scenarios.

B. *Policy Based Management for Wireless Sensor Networks*

ESCAPE [8] is a policy framework for programming sensor network applications. ESCAPE starts from the *Separations of Concerns* principle, in which developers use policies to exclusively specify interactions between components, removing interaction code from these individual components. Unfortunately, ESCAPE is limited to the expression of non-functional concerns and cannot cope with dynamic system evolution or adaptation.

Driven by industry initiatives, the Service Component Architecture (SCA) defined a Policy framework specication [9], which aims to use policies for describing capabilities and constraints that can be applied to service components or to the interactions between different service components. While not being bound to a specific implementation technology, the SCA policy framework focusses on service-oriented environments which require more resources than are available in embedded environments.

C. *Software Deployment Approaches for Wireless Sensor Networks*

Macro-programming approaches, such as Magnet OS [10] and RuleCaster [11], compile a network level program into smaller programs that can run on a single node. These node level programs are then assigned to nodes by the system. If assignment based on node capabilities yields multiple possibilities, these approaches perform optimization of a certain system property. While this approach may be optimal for single-application wireless sensor networks, no support is provided for multi-application sensor networks.

Semantic Streams [12] is a framework that allows the developer to pose declarative queries over semantic compositions of sensor streams. The framework allows user specified cost functions to be attached to queries. The planner uses this cost function to choose between possible execution scenarios for the query. Semantic Streams thus allows users to influence quantitative trade offs made by the system. However, this affects only the querying of pre-deployed functionality and provides no support for the evolution or adaptation of system functionality.

MiLAN [13] also addresses the topic of determining the optimal subset of sensors that is required to reach a desired quality-of-service level. MiLAN confirms that there are usually multiple subsets of sensors that can provide the required data, but with different quality-of-service properties. We draw inspiration from this approach, adding flexible specification of the deployment environment and quality metrics, as described in [3].

III. The DISSENT Middlewre

This section provides a brief overview of the supporting technologies provided in the DisSeNT middleware framework. Generic component development is supported by the Loosely-coupled Component Infrastructure (LooCI). Policy-based management support for deployed system is provided by the Policy-based Management Architecture (PMA). Quality aware software deployment and reconfiguration support is provided by the Quality Aware Reconfiguration Infrastrutcure (QARI). A brief overview of each of these technologies is provided in sections 3.1 – 3.3 respectively. For a more complete description of each of these technologies, we refer the reader to [1], [2] and [3] respectively.

A. *The Loosely-coupled Component Infrastructure (LooCI)*

LooCI [1] is a run-time reconfigurable component and binding model that is designed for networked embedded systems such as WSNs. LooCI has a small footprint and offers good performance even on embedded sensor nodes. In the LooCI component model an interface is modelled by the publication of a specific event type on the event bus, while a receptacle is created by the subscription of a component to a specific type of event. The LooCI component model is platform and language independent, with ports available for the Java-based Sun SPOT platform, the Contiki-based AVR Raven platform as well as standard PCs. LooCI supports runtime reconfiguration including: (i.) dynamic component deployment, (ii.) distributed mechanisms to start/stop a component and to place components into quiescent state and (iii.) support for the dynamic modification of bindings between components. All interactions between LooCI componenents occur via the exchange of hierarchical and globally typed events that are exchanged over a distributed event bus. A more detailed description of LooCI can be found in [1].

B. *The Policy-based Management Architecture (PMA)*

PMA [2] integrates with the LooCI event bus in order to provide support for tailoring and management of system functionality through the use of high level policies. As all interactions between LooCI components occur over the event bus, it is possible to tailor any aspect of system functionality by modifying the manner in which events are allowed to propagate. To facilitate this type of application management, the LooCI runtime is extended with a compact policy engine, which executes a simple event-condition-action policy specification language. The ECA policy specification language offered by PMA is a good fit with the typical skill-set of a system administrator and is supported by a simple back-end tool that allows policies to be applied at various levels of granularity including network-wide, per-mote, per-component or even per-interface. A more detailed description of PMA can be found in [2].

C. The Quality Aware Reconfiguration Infrastructure (QARI)

Software deployment is achieved using QARI [3]. Following application composition, quality aware deployment specification is achieved using two high-level domain-specific languages. QARI specifications contain two parts:

- A *target network specification* is provided using the high level network description language of QARI. This language allows the application developer to specify regions, which may be nested and to map nodes onto these regions.
- A *quality aware deployment specification* is provided that describes how software should be deployed within the regions created in the first stage. This simplifies the deployment process and allows for the automatic repair of compositions at run-time.

After deployment, the QARI runtime will poll the network using the reflective functionality of LooCI [1] and compare observed component availability against the required quality aware deployment specification. Where a region is found not to meet the requirements defined in the associated quality specification, repair action will be undertaken. If it is not possible to repair the deployment to meet the provided quality specification, then QARI will disseminate an exception to the actor who is responsible for the deployment.

IV. Application Composition

This section describes how the LooCI component model was used in conjunction with PMA to realize the river monitoring application. In brief, this application allows for central monitoring and alert dissemination when water levels (measured using a pressure sensor) or pollution levels (measured using conductivity and methane sensors) rise above acceptable norms. Tamper detection is implemented using a three dimensional accelerometer. The Java-based Sun SPOT sensing platform was used in all experiments. The complete application composition is shown in Figure 1 below.

A. Development of a Component Composition for River Monitoring

The river monitoring composition consists of five components:

- *ConductivitySensor:* the ConductivitySensor component exposes readings from the water conductivity sensor via a single interface of type CONDUCT.
- *MethaneSensor:* the MethaneSensor component exposes readings from the CH4 sensor via a single interface of type METHANE.
- *AccelerometerSensor:* the AccelerometerSensor component exposes readings from the Sun SPOT accelerometer via a single interface of type ACCEL.
- *PressureSensor:* the PressureSensor component exposes readings from the vented gauge hydrostatic level sensor via a single interface of type PRESSURE.
- *EnvironmentalHazardAlert:* the EnvironmentalHazardAlert component offers a single receptacle of type SENSOR which displays environmental sensor readings in a graphical user interface. Due to the hierarchical properties of the event ontology this component will display readings from all deployed sensing components.

B. Policy Based Augmentation of the Component Based Composition

The river monitoring application composition includes four policies:

- *ConductivityDetectionPolicy:* this policy effectively acts as a filter, allowing CONDUCTIVITY events to pass only when they rise above a specified alert threshold.
- *MethaneDetectionPolicy:* this policy filters METHANE events, only allowing them to pass where they exceed the specified threshold for methane gas emissions.
- *TamperDetectionPolicy:* this policy filters ACCEL readings, only allowing them to pass to the EnviornmentalHazardAlertComponent where they exceed a threshold that indicates theft or vandalism.
- *FloodDetectionPolicy:* this policy filters PRESSURE readings, only allowing them to pass to the EnviornmentalHazardAlertComponent where they exceed a threshold that indicates flooding.

Components are wired as shown in Figure 1. Lines ending in balls denote interfaces, while lines ending in cups denote receptacles. The number above each interface/receptacle indicates its type in the global LooCI event type system.

As can be seen from the composition shown in Figure 1, policies are used to intercept and filter the output of each component. This allows the developer to specify alert levels in a simple and high-level fashion.

Components are well suited to the encapsulation of generic re-usable functionality as they provide concrete interfaces that are easy to re-use in future applications.

Policies are well suited to encapsulating dynamic concerns that change rapidly, as they are specified using a high level language and, as they are compact, policy updates entail lower overhead.

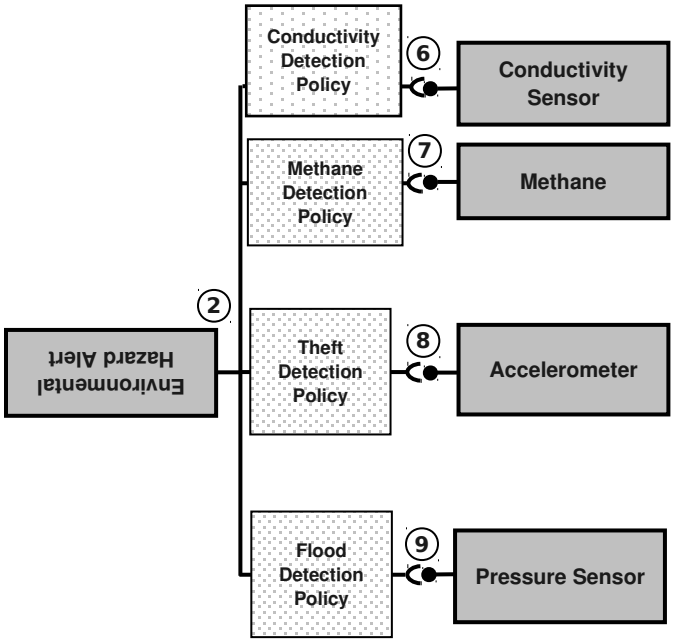

Figure 1. Multi-Paradigm composition

V. Evaluation

We have evaluated the DisSeNT middleware through analysis of the case-study application described in Section 4. We first evaluated the efficiency of our implementation in terms of code footprint and development overhead using lines of code analysis. All experiments were performed using the SunSPOT/Java version of LooCI.

A. Code Footprint

For resource constrained embedded systems, maintaining a minimal application footprint is critical for two primary reasons: Firstly, a small application footprint leaves more space available for data logging. Secondly, in the case of reconfigurable systems, where component updates may be transmitted over the air, a smaller component footprint reduces radio activity and thus conserves battery life.

The static memory footprint of all components is shown in Table 1. The static memory footprint of all policies is shown in Table 2 below. As can be seen from the table, all components have a footprint of less than 2kB, with the exception of The *EnvironmentalHazardAlert* component that runs on the resource-rich back-end (and therefore its larger size has no significant impact on system performance). All policies consuming between 94 bytes and 95 bytes – an order of magnitude less than components. This is particularly advantageous as policies are used to represent dynamic concerns such as alert thresholds and are therefore more likely to be updated at run-time over-the-air.

TABLE I. Component Footprint

	Size
ConductivitySensor:	1.8 kB
MethaneSensor:	1.7 kB
PressureSensor:	1.7 kB
AccelerometerSensor:	1.9 kB
EnvironmentalHazardAlert:	13.2 kB
Total:	20.3 kB

TABLE II. Policy Footprint

	Size
ConductivityDetectionPolicy:	94 bytes
MethaneDetectionPolicy:	94 bytes
TamperDetectionPolicy:	95 bytes
FloodDetectionPolicy:	94 bytes
Total:	377 bytes

B. Development Overhead

This section provides a quantitative assessment of development overhead. Table 3 and Table 4 respectively provide a Lines of Code (LoC) analysis for all components and policies in our application. As can be seen from Table 3, LooCI components are relatively compact and impose limited overhead on developers in terms of LooCI-specific code. As shown in Table 4, the high-level nature of PMA policies allows for even more compact expression of rich functional and non-functional objectives.

TABLE III. Lines of Code: Components

	Functional Code	LooCI Code
ConductivitySensor:	48	11
MethaneSensor:	48	11
PressureSensor:	48	11
AccelerometerSensor:	42	11
EnvironmentalHazardAlert*:	41	23
Total:	227	67

TABLE IV. Lines of Code: Policies

	Size
ConductivityDetectionPolicy:	10
MethaneDetectionPolicy	10
TamperDetectionPolicy:	10
FloodDetectionPolicy:	10
Total:	40

While it is difficult to access developer overhead through LoC analysis alone, we believe that the results obtained above are fair and representative.

VI. Conclusions and Further Work

This paper has demonstrated how the multi-paradigm approach embodied by DisSeNT can be used to realize efficient software for wireless sensor networks. Through a

real-world evaluation in the context of WSN-based we have shown that the resulting software is efficient and extensible.

Our future work will focus on realising a large scale deployment of motes running the DisSeNT middleware, which will allow us to test our approach to WSN software development at large scale. In these large scale deployments we will also test the Contiki and OSGi ports of LooCI.

ACKNOWLEDGEMENTS:

This research is conducted in the context of the IWT-STADIUM project No. 80037 [4]. Wouter Horré is a PhD fellow of the Research Foundation – Flanders (FWO). Dr. Jo Ueyama and Dr. Danny Hughes would like to thank FAPESP for funding the case-study elements of this research project (Process ID 08/05346-4 and 09/01881-5).

REFERENCES

[1] Hughes D., Thoelen K., Horré W., Matthys N., Michiels S., Huygens C., Joosen W., LooCI: A Loosely-coupled Component Infrastructure for Networked Embedded Systems, in the proceedings of the 7th International Conference on Advances in Mobile Computing & Multimedia (MoMM'09), December 2009, ACM.

[2] Matthys N., Huygens C., Hughes D., Michiels S., Joosen W., Flexible Integration of Data Qualities in Wireless Sensor Networks, in the proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks (MidSens'09), Urbana Champaign, Illinois, USA, December 2009.

[3] Horré W., Hughes D., Michiels S., Joosen W., QARI: Quality Aware Software Deployment for Wireless Sensor Networks, in the proc. of the 7th International Conference on Information Technology : New Generations (ITNG'10).

[4] IWT Stadium project 80037, software technology for adaptable distributed middleware: http://distrinet.cs.kuleuven.be/projects/stadium/

[5] Gay D., Levis P., Von Behren R., Welsh M., Brewer E., Culler D., The NesC Language: A Holistic Approach to Networked Embedded Systems, in Proc. of the conference on Programming Language Design and Implementation, ACM SIGPLAN 2003, San Diego, California, USA, pp. 1-11.

[6] Coulson G., Blair G., Grace P., Taiani F., Joolia A., Lee K., Ueyama J. and Sivaharan T, A Generic Component Model for Building Systems Software, in ACM Transactions on Computer Systems, Vol. 26, No. 1, Feb 2008.

[7] Costa P., Coulson G., Gold R., Lad M., Mascolo C., Mottola L., Picco G.P., Sivaharan T., Weerasinghe N., Zachariadis S., The RUNES Middleware for Networked Embedded Systems and its Application in a Disaster Management Scenario, in Proc. of the 5th Annual IEEE International Conference on Pervasive Computing and Comunications (PerCom'07), White Plains, New York, March 2007, pp. 69–78.

[8] Russello G., Mostarda L., Dulay N., Escape: A component-based policy framework for sense and react applications, in the proc. of the 11th International Symposium on Component-Based Software Engineering, 2008, pp. 212-229.

[9] OSOA, SCA Policy Framework. SCA v1.0, March 2007, avaialable online at: http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf

[10] Liu H., Roeder T., Walsh K., Barr R., Sirer E. G., Design and implementation of a single system image operating system for ad hoc networks, in proc. of the 3rd international conference on Mobile systems, applications, and services. New York, NY, USA, 2005, pp. 149–162.

[11] Bischoff U., Kortuem G., Rulecaster: A macroprogramming system for sensor networks, in proc. of the OOPSLA Workshop on Building Software for Sensor Networks, Oct. 2006.

[12] K. Whitehouse, F. Zhao, and J. Liu, "Semantic streams: A framework for composable semantic interpretation of sensor data." in *EWSN*, 2006, pp. 5–20.

[13] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications." *IEEE Network*, vol. 18, no. 1, pp. 6–14, 2004.

[14] Simon D., Cifuentes C., Cleal D., Daniels J., White D., Java on the Bare Metal of Wireless Sensor Devices: the Squawk Java Virtual Ma chine, in Proc. of the 2nd International Conference on Virtual Execution Environments, Ottawa, Canada, June 2006, pp 78 – 88.

[15] Huygens C., Hughes D., Legaisse B., Joosen W., Streamlining Development for Networked Embedded Systems Using Multiple Paradigms, to appear in IEEE Software, special edition on Multi-Paradigm Programming, IEEE